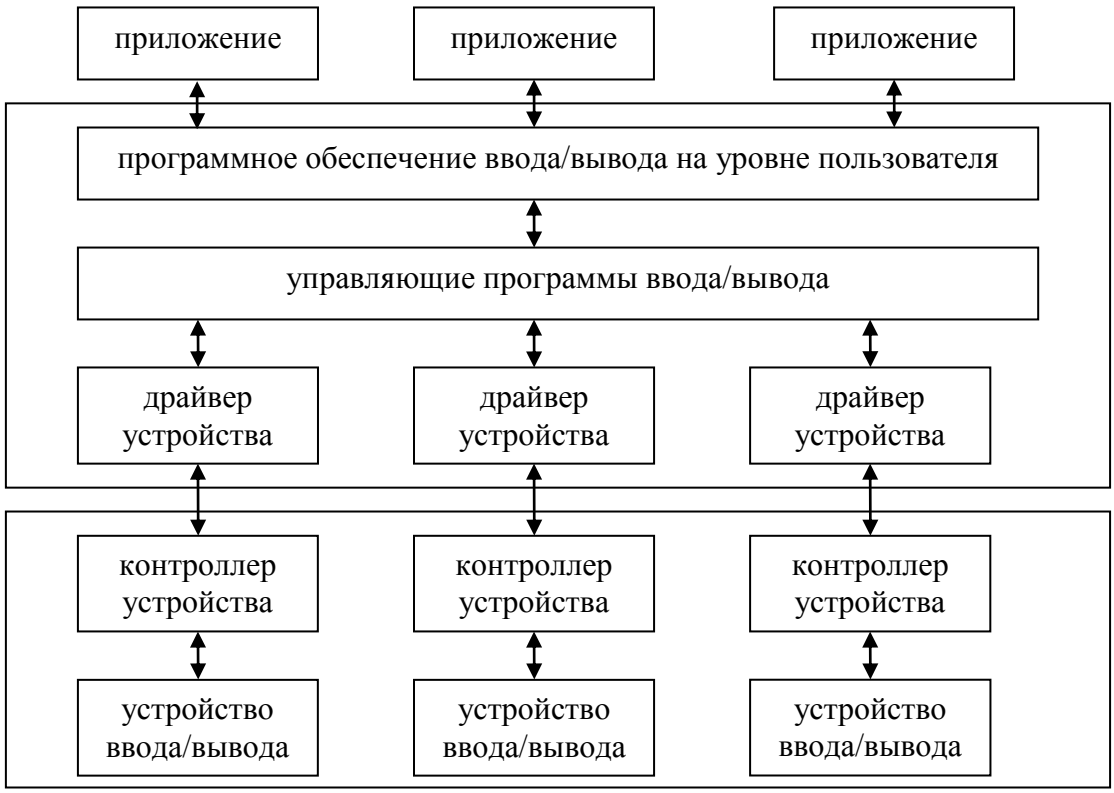
12. Управление внешними устройствами

Подсистема ввода/вывода данных – это комплекс технических и программных средств, используемых для взаимодействия ВС с внешним миром. За время существования вычислительной техники было создано множество разнообразных устройств ввода/вывода со своими параметрами и характеристиками. Более того, устройство одного и того же типа (например, лазерный принтер), как правило, выпускается в разных модификациях. Совершенно очевидно, что прикладным программам не обязательно знать о конкретном используемом виде устройства, поэтому важнейшая задача подсистемы ввода/вывода – это экранирование приложений от технических особенностей используемых устройств.

По этой причине подсистема ввода/вывода в современных ОС строится по многоуровневому принципу:

- нижний (аппаратный) слой включает два уровня:
    - сами устройства ввода/вывода;
    - электронные схемы для управления этими устройствами (контроллеры);
- верхний (программный) слой включает следующие уровни:
    - специализированные программы-драйверы для непосредственного управления устройствами;
    - независимые от устройств управляющие программы ядра системы (диспетчер ввода/вывода);
    - программное обеспечение ввода/вывода на уровне пользователя, т.е. высокоуровневые системные вызовы для операций ввода/вывода.

Далее каждый из уровней рассматривается более подробно.



К основным устройствам ввода/вывода относятся: клавиатура, мышь, различные манипуляторы, монитор, принтер, сканер, цифровой фотоаппарат, видеокамера, модем, диски разных типов, сетевые платы и т.д. Все они имеют разные характеристики и могут классифицироваться по разным признакам.

Прежде всего, устройства ввода/вывода достаточно условно можно разбить на две большие группы – **символьные** и **блочные.** Символьные устройства обрабатывают последовательность (поток) байтов строго друг за другом. Блочные устройства позволяют группировать соседние байты в блоки с возможностью адресации каждого блока в отдельности, независимо от других блоков. Типичный пример – дисковые накопители.

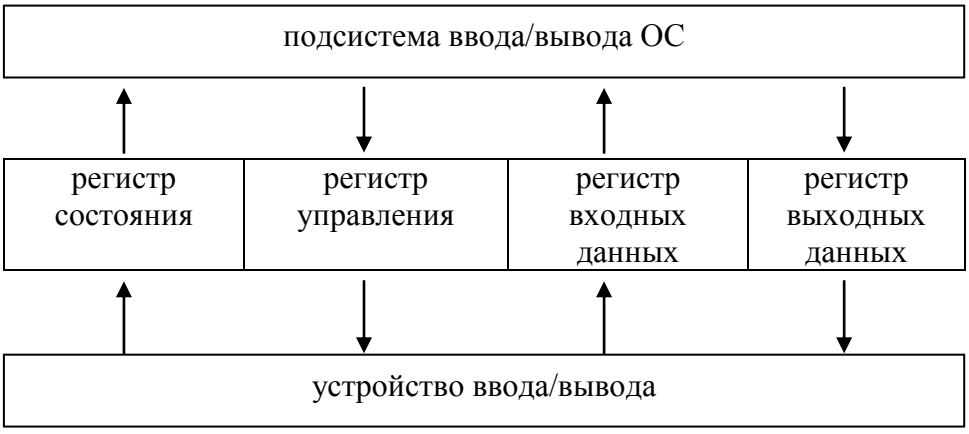
Другой очевидной классификацией устройств является их специализация – одни устройства предназначены только для ввода данных, другие – только для вывода, третьи могут выполнять оба действия.

Непосредственное управление устройствами выполняют электронные схемы-**контроллеры.** Несмотря на их многообразие, есть ряд принципов,

общих для всех контроллеров. Важнейшее свойство контроллеров – это наличие специальных **регистров,** через которые организуется все общение операционной системы с устройствами. Как правило, регистры специализируются в зависимости от их назначения следующим образом:

- регистр **состояния** содержит биты, определяющие текущее состояние устройства (свободно/занято, готово к выполнению команды, успешное или ошибочное завершение команды); биты регистра состояния формируются самим устройством и могут лишь читаться программами ОС;
- **управляющий** регистр содержит очередную команду, которую должно выполнить устройство (занести байт(ы) в регистр данных, взять байт(ы) из регистра данных и обработать его);
- регистр **входных** данных, в который устройство помещает очередной байт (или несколько байтов) данных и откуда этот байт (байты) считывается системой;
- регистр **выходных** данных, в который система помещает очередной байт (байты) выводимых данных для обработки его устройством.

Разрядность регистров определяется разрядностью соответствующей шины. Если для первых двух регистров достаточно одного-двух байтов, то данные могут передаваться группами по 4 байта.



Каждому регистру каждого установленного в системе устройства присваивается порядковый номер-адрес, для обозначения которого используется термин **“порт ввода/вывода”**. Набор используемых в системе портов образует **адресное пространство** ввода/вывода. Это адресное пространство может либо занимать часть адресов основного адресного пространства, либо существовать отдельно от него. Часто используется комбинированный подход – адреса портов состояния и управления реализуются как самостоятельное адресное пространство, а порты для непосредственного обмена данными занимают часть основной памяти.

Для взаимодействия с портами ввода/вывода в систему команд часто вводятся специальные команды ввода/вывода:

- записать один-два байта в порт с заданным номером (команда OUT);
- прочитать один-два байта из порта с заданным номером (команда IN).

Эти команды относятся к группе **привилегированных,** т.е. они могут выполняться только в режиме ядра. Попытка выполнения этих команд внутри прикладной программы, работающей в пользовательском режиме, приводит к аварийному прерыванию выполнения программы. Такой подход позволяет изолировать приложения от непосредственного общения с устройствами ввода/вывода. Только программы ядра системы имеют право напрямую общаться с устройствами, тогда как приложения для данных целей должны использовать специальные системные вызовы. Это особенно важно для многозадачных ОС, где централизованное управление всеми ресурсами (к которым относятся и устройства ввода/вывода) является чрезвычайно важной задачей.

Программирование операций ввода/вывода на уровне команд представляет собой достаточно сложную задачу, решение которой требует знания тонкостей работы конкретного устройства. Поэтому в подсистему ввода/вывода включается специальный уровень **программ-драйверов,** как правило, создаваемых фирмами − разработчиками аппаратуры. Набор

драйверов – это важнейшая часть подсистемы ввода/вывода и операционной системы в целом. Для устранения возможного разнобоя при написании драйверов используются специальные правила, четко регламентирующие процесс взаимодействия драйверов как с ядром системы, так и с контроллерами устройств. Например, для написания драйверов для ОС семейства Windows корпорация Microsoft разработала специальный стандарт Windows Driver Model (WDM). Для облегчения написания драйверов в соответствии с этим стандартом разработан целый ряд программных инструментов (DDK – Driver Development Kit).

В целом, каждый драйвер должен предоставлять четко определенный набор функций, которые могут вызываться программами более высокого уровня. Типичный алгоритм взаимодействия драйвера с устройством и с вышележащим программным уровнем включает следующие шаги:

1. Драйвер получает от управляющего уровня команду на выполнение операции ввода/вывода.
2. Драйвер записывает в управляющий порт контроллера сигнал “начать операцию”.
3. В ответ на это устройство устанавливает в регистре состояния соответствующий признак возможности выполнения операции.
4. Драйвер проверяет значение этого признака и в зависимости от результата либо продолжает процесс взаимодействия, либо прекращает его с выдачей сообщения на более высокий уровень.
5. Если операция возможна, драйвер записывает в управляющий порт код операции.
6. Если выполняется операция вывода, то драйвер в порт выходных данных записывает и сам выводимый элемент данных.
7. Когда устройство закончит обработку элемента данных, то оно в порт состояния помещает признак успешного завершения, а при вводе еще и заносит во входной порт элемент данных.

8. Драйвер проверяет код завершения операции и в зависимости от его значения либо повторяет цикл ввода/вывода для следующего элемента данных, либо передает на верхний уровень сообщение о возникшей ошибке.

Эта общая схема требует уточнения для конкретных операций. Например, вывод на принтер по одному элементу данных неэффективен, поэтому в схему добавляются следующие шаги:

- стандартная программа печати накапливает группу выводимых элементов данных в промежуточном буфере вывода и передает адрес этого буфера драйверу принтера;
- драйвер принтера запускает процесс печати и блокируется на время выполнения операции вывода этой порции данных (шаги 1-6 приведенного выше алгоритма);
- когда принтер закончит печать заданной порции данных, он генерирует **прерывание** (механизм прерываний рассматривается ниже), по которому драйвер принтера вновь активизируется и выполняет завершающую работу (шаги 7 и 8).

Необходимо помнить, что драйвер – это программа, для выполнения которой ей необходимо предоставить процессор, а поскольку большинство операций ввода/вывода выполняется относительно медленно, то блокирование драйвера на время выполнения устройством операции ввода/вывода вполне оправданно.

Большинство драйверов имеют следующую типовую структуру:

- секция запуска предназначена для инициирования операции ввода/вывода;
- секция продолжения выполняет основную работу по обмену данными, в том числе такую важную, как обработка прерываний с запуском соответствующей компоненты секции продолжения; часто драйверы содержат несколько секций продолжения;
- секция завершения выполняет необходимые завершающие действия.

Одним из фундаментальных понятий подсистемы ввода/вывода является понятие прерывания, важность которого для функционирования вычислительных систем настолько высока, что требует отдельного рассмотрения.

Прерывание (interrupt) – это механизм, с помощью которого вычислительная система может реагировать на возникающие в ней события. Система должна уметь распознать возникшее событие и обработать его, т.е. выполнить предусмотренные для этого действия. При этом процессор должен переключиться на выполнение программного кода обработки прерывания. Важнейшая особенность прерываний – непредсказуемость моментов их возникновения.

Основной тип прерываний – это **внешние** или **аппаратные** прерывания, источниками которых являются контроллеры различных устройств, в том числе - устройств ввода/вывода. Примерами таких прерываний являются: нажатие клавиши на клавиатуре, нажатие кнопки мыши, завершение дисковой операции, сигнал от системного таймера и т.д. Именно эти прерывания позволяют организовать согласованную работу всех устройств вычислительной системы.

Интуитивно понятно, что возникающие в системе события могут иметь разную степень важности, поэтому механизм прерываний реализует **приоритетную** схему обработки. Все прерывания упорядочены по степени убывания важности следующим образом:

- критические сбои аппаратуры (сбой питания, ошибка шины);
- прерывания от системного таймера (основа механизма квантования при вытесняющей многозадачности);
- прерывания от дисковых устройств;
- прерывания от сетевых устройств;
- прерывания от клавиатуры и мыши.

Эти приоритеты играют большую роль в алгоритме обработки прерываний. Необходимо отметить, что обработка прерываний в

существенной степени выполняется на аппаратном уровне, что объясняется большой частотой их возникновения и необходимостью обработки с максимально высокой скоростью. Для этого многие современные вычислительные системы имеют специальное устройство – **контроллер прерываний.** Каждое устройство получает свой **номер** прерывания (иногда один и тот же номер могут иметь сразу несколько устройств). Когда устройство хочет сообщить о некотором событии, оно выставляет сигнал на выделенную ему линию шины. Этот сигнал распознается контроллером прерываний, который с помощью своей встроенной логики реализует следующие действия:

- если процессор занят обработкой более приоритетного прерывания, то новое прерывание игнорируется, не обрабатывается, но устройство − источник прерывания продолжает удерживать сигнал прерывания на шине до наступления “лучших времен”, т.е. освобождения процессора;
- если процессор выполняет менее приоритетную работу, эта работа прерывается с сохранением контекста и происходит переключение на обработку нового прерывания.

Простейший способ запуска обработчиков прерываний состоит в использовании специальных таблиц, называемых **таблицами векторов прерываний.** Такая таблица содержит начальные адреса размещения в памяти соответствующих обработчиков прерываний. Сама таблица находится в памяти в строго фиксированном месте и индексируется номером прерывания. Например, если прерывания перенумерованы от 0 до 255, то таблица содержит 256 записей по 4 байта, т.е. занимает 1 Кб памяти. В этом случае по номеру прерывания (индекс в таблице!) сразу определяется адрес той команды, с которой начинается код соответствующего обработчика. Этот адрес заносится в счетчик команд, и тем самым процессор переключается на выполнение кода обработчика. Само переключение контекста выполняется обычным образом, с сохранением информации о прерванном потоке и с его возобновлением, когда это будет возможно.

Кроме внешних (аппаратных) прерываний часто рассматриваются и два других типа прерываний – **внутренние** и **программные.** Внутренние прерывания возникают при выполнении процессором потока команд, когда очередная команда в этом потоке по тем или иным причинам не может быть выполнена. Например: попытка деления на ноль, обращение по неправильному адресу, попытка выполнить запрещенную команду, отсутствие запрошенной страницы памяти и т.д. Такие особые ситуации (часто их называют **исключениями**) распознаются процессором и должны быть соответствующим образом обработаны. Программные прерывания, или **псевдопрерывания,** генерируются специальными командами и часто используются программистами для быстрого вызова важных системных функций. Программные прерывания имеют меньший приоритет по сравнению с аппаратными и поэтому обрабатываются в последнюю очередь.

В качестве примера ниже приводятся шестнадцатеричные адреса некоторых прерываний процессоров семейства Intel.

Аппаратные прерывания:

- 08h - от системного таймера;
- 09h - от клавиатуры;
- 0Eh - от гибкого диска;
- 76h - от жесткого диска.

Внутренние прерывания:

- 00h - деление на 0;
- 0lh - пошаговое выполнение команд программы в режиме отладки.

Программные прерывания:

- 10h – обращение к подпрограммам BIOS управления видеосистемой;
- 13h – обращение к подпрограммам BIOS управления дисками;
- 16h – обращение к подпрограммам BIOS управления клавиатурой.

Управляющие программы, которые можно назвать одним термином **“диспетчер (или менеджер) ввода/вывода”**, реализуют верхний уровень программного слоя подсистемы ввода/вывода. Главное, но не единственное

назначение диспетчера – принимать вызовы на выполнение операций ввода/вывода от прикладных и системных процессов, транслировать эти вызовы соответствующим драйверам и возвращать результаты выполнения затребованных операций процессам-потребителям. Особенно актуальной эта задача становится в условиях многозадачности, когда один и тот же ресурс может потребоваться нескольким процессам.

Для решения такой задачи диспетчер ввода/вывода должен в каждый момент времени знать текущее состояние каждого из установленных в системе устройств. Для этого система создает и поддерживает одну или несколько связанных информационных структур данных (таблиц). Основная таблица должна содержать перечень всего имеющегося в системе оборудования. Для каждого устройства создается своя **запись-дескриптор,** содержащая следующие данные:

- тип устройства и его основные характеристики;
- номера используемых портов и выделенных устройству прерываний;
- информация о драйвере устройства, в частности – адреса точек входа всех входящих в него процедур;
- текущее состояние устройства и, в частности, идентификатор процесса, использующего устройство в данный момент;
- если устройство использует буферы для обмена данными, то адреса размещения в памяти всех буферов;
- адрес начала списка процессов, ожидающих данное устройство;
- логическое имя устройства, используемое высокоуровневыми вызовами.

Обобщенный алгоритм работы диспетчера ввода/вывода можно представить следующим образом:

- диспетчер принимает запрос на выполнение операции в виде системного вызова от активного процесса и проверяет правильность передаваемых в этом запросе параметров;

- проверяется состояние затребованного устройства, и если устройство занято, то соответствующий процесс ставится в очередь ожидания с изменением его состояния и запуском планировщика потоков;
- если устройство свободно, то диспетчер:
    - вызывает соответствующий драйвер, передавая ему всю необходимую информацию;
    - переводит (если это необходимо) процесс-источник запроса в состояние ожидания;
    - “засыпает” до тех пор, пока драйвер не завершит выполнение операции;
- после выполнения драйвером необходимых действий, диспетчер вновь активизируется, анализирует полученную от драйвера информацию и передает результат операции процессу−источнику запроса, изменяя, возможно, его состояние.

В итоге, подобный подход позволяет загружать процессор операциями ввода/вывода только в случае необходимости, освобождая его для других задач на время работы относительно медленных устройств ввода/вывода. Еще большего эффекта удается достичь за счет использования специального механизма **прямого доступа к памяти** (DMA – Direct Memory Access). Этот механизм используется при передаче больших объемов данных непосредственно между основной памятью и устройством ввода/вывода, например – диском. При этом процессор из самого обмена исключается и может заниматься другой работой. Управление процессом обмена берет на себя контроллер DMA, который можно рассматривать как вспомогательный сопроцессор, работающий параллельно с основным процессором.